Multipath-TCP in Network Simulator 3

Morteza Kheirkhah, Ian Wakeman and George Parisis
Department of Informatics, University of Sussex, UK
{m.kheirkhah, ianw, g.parisis}@sussex.ac.uk

ABSTRACT

In this paper we present our work on designing and implementing an NS3 model for MultiPath TCP (MPTCP). Our MPTCP model closely follows MPTCP specifications, as described in RFC 6824, and supports TCP NewReno loss recovery on a per subflow basis. Subflow management is based on MPTCP's kernel implementation. We briefly describe how we integrate our MPTCP model with NS3 and present example simulation results to showcase its working state.

1. INTRODUCTION

MultiPath TCP (MPTCP) has been gaining significant attention during the past years. Its ability to spread network traffic to multiple subflows of a single MPTCP connection, in a way that fairness with other competing flows is preserved, promises more efficient usage of network resources and resilience in the face of network failures. In data centres, multi-homed servers, network path multiplicity and very high aggregated bandwidth are becoming the norm and transport protocols that could maximise the usage of network resources and minimise flows' completion time, while being fair with existing TCP flows, are heavily researched. Network simulations are the only realistic route to capture, examine and understand the behaviour of a transport protocol at the scale required in the context of data centres. Existing research is commonly based on custom-developed network simulators, instead of relying on and extending existing, well-tested, open-source code. This makes verifying the correctness of the simulation itself extremely difficult. Software bugs, even minor ones, can seriously affect the quality and reliability of results. In many cases, simulators' source code is not made available and, therefore, it is impossible to repeat and verify published results.

To date, there has been a single attempt to implement MPTCP in NS3 version 3.6 [2]. However, this model was never merged with any stable version of NS3 and also became obsolete after TCP was rewritten in NS3.8. In this model only a single client could connect to an MPTCP server; i.e. a server could not fork new MPTCP connections. This is a problem particularly when dealing with realistic traffic models in data centres. Additionally, the model did not support nodes running TCP and MPTCP in parallel, a feature that is often required in data centre experimentation, when evaluating fairness among competing TCP and MPTCP flows. MPTCP tokens [3], which uniquely identify MPTCP connections in a host and are used to associate new subflows to an existing MPTCP connection, were not used either. Finally, several other simplifications (e.g. The MPTCP connection and its subflows do not follow standard TCP state transitions) were present.

In this paper we present an NS3 model of MPTCP, which is based on RFC 6824 [3]. Subflow management is done similarly to the Linux kernel implementation and loss recovery is based on TCP NewReno. Our model overcomes all limitations described above with the ambition to become the official MPTCP model for NS3. The source code will soon be submitted for review.

2. MPTCP MODEL

2.1 Connection and SubFlow Management

Our MPTCP model follows the Linux kernel implementation of MPTCP [1]. Each MPTCP connection can have several subflows, each of which operates as a regular TCP connection. Each connection starts with a master subflow, the only subflow presented to the application through the exported socket interface. In our model we have implemented the following classes:

MpTcpSocketBase. This class implements the MPTCP control block and exports the socket API to NS3 applications. It performs data scheduling, packet reordering, congestion control and loss recovery for all MPTCP subflows. On the server side, a MPTCP connection is represented with one listening *MpTcpSocketBase* object which forks new *MpTcpSocketBase* objects for each accepted MPTCP connection. On the client side an *MpTcpSocketBase* object represents an MPTCP connection with a server. *MpTcpSocketBase* is a subclass of *TcpSocketBase*.

MpTcpSubflow. This class represents an MPTCP subflow and is a subclass of the *Object* class. An *MpTcpSocketBase* object may have multiple *MpTcpSubflow* objects.

TcpL4Protocol. We have changed *TcpL4Protocol* so that MPTCP connections can be handled, without disrupting any existing TCP functionality. As with single-path TCP models, this class is an interface between the transport and network layers and is responsible for sending and receiving packets to and from the network layer, respectively. When a packet is received, it looks up for an *Ipv4EndPoint* based on the TCP header's four-tuple. In our model, several *Ipv4EndPoint* objects, representing endpoints for respective MPTCP subflows, can be associated to one MPTCP connection. MPTCP token support is implemented in this class as well. A token is a locally unique identifier assigned to an MPTCP connection upon establishment. When a sender initiates a new subflow, the receiver looks up for an

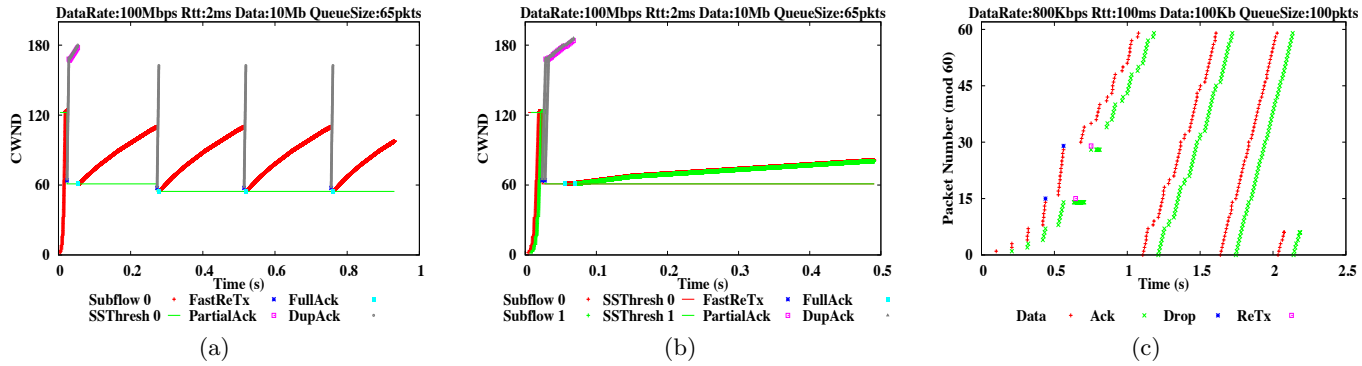

Figure 1: (a) MPTCP with a single subflow, (b) MPTCP with two subflows (c) MPTCP with a single subflow and two packet drops (as in [4]). Packet size is 536 bytes in above simulations

Ipv4EndPoint based on the token passed in the MP-JOIN option and forwards the request to the respective *MpTcpSocketBase* object. Requests for new MPTCP connections are resolved using the four-tuple and forwarded to the listening *MpTcpSocketBase* object.

2.2 MPTCP Signalling

Our MPTCP model closely follows the specifications set in RFC 6824[3]. MPTCP signalling is implemented as follows:

Connection Establishment. MPTCP connection establishment follows the standard TCP three-way handshake. The client attaches an MP-CAPABLE option in the SYN packet to denote its MPTCP support. The SYN packet is forwarded to a listening *MpTcpSocketBase*, in turn a new *MpTcpSocketBase* is forked to continue communication with the client. If the server supports MPTCP, an MP-CAPABLE option is attached to the SYN-ACK response. A randomly generated 32-bits token is carried in the MP-CAPABLE option. The token is mapped to the respective *MpTcpSocketBase*; this mapping being stored in the *TcpL4Protocol* object.

Subflow Establishment. After connection establishment, communicating endpoints advertise available IP addresses to each other using the ADD-ADDR option. New subflows can be later established using a three-way handshake and attaching the MP-JOIN option in the SYN packet. The token is also attached so that the receiving side can resolve the subflow establishment request to an existing MPTCP connection, as described in the previous section.

Sending and Receiving Data Packets. MPTCP utilises two separate sequence number spaces, one per-connection (64-bits) and one per-subflow (32-bits). The former is used for packet reordering and loss recovery at connection level and it is signalled via the Data Sequence Signal (DSS) option. The latter is used for the same reasons at a subflow level and is carried in the sequence number field of the TCP header.

Connection Teardown. As with standard TCP, each subflow terminates with a four way FIN handshake. An MPTCP connection is also terminated at a connection level by signalling a DATA-FIN option. In our model, connection teardown is performed at a subflow level. MPTCP endpoints deallocate all resources (the token mapping and *MpTcpSocketBase* object are also deleted from the *TcpL4Protocol* object) only when all subflows have been closed.

3. CURRENT STATE AND FUTURE WORK

Our MPTCP model is in a fully working state. A few minor additions are still required to fully comply with the RFC 6824 (e.g. MP-CAPABLE does not exchange keys for securing a connection and MP-PRIO is not implemented). To showcase our model we present results from some simple simulation scenarios. Figure 1(a) illustrates the evolution of the congestion window when an MPTCP connection with a single subflow is opened between two nodes connected via a point-to-point link. For clarity, we haven't plotted the results from simulations using the standard TCP NewReno model; they were identical to the ones presented in the figure. In Figure 1(b) we plot the congestion window progression, when two subflows are used via two point-to-point links. We have also examined the behaviour of our NewReno loss recovery mechanism by reconstructing a simulation scenario from [4]. Our results in 1(c) are very similar to Figure 3 in [4].

This work is part of our larger effort towards building an NS3-based simulation platform for experimenting with transport protocols and congestion control algorithms in data centres. We plan to make necessary changes in the core NS3 code so that congestion control and loss recovery mechanisms can be applied to different transport protocols by reusing existing code. For example we had to rewrite the NewReno loss recovery code in our *MpTcpSocketBase* class, although we could have reused code from the *TcpNewReno* class. A major challenge towards building a simulation platform for data centres is the scale of simulations. We are currently working on identifying the current limits and the required changes to scale up to large simulated topologies.

4. REFERENCES

[1] S. Barré, C. Paasch, and O. Bonaventure. MultiPath TCP: From Theory to Practice. In *Proc. of IFIP Networking*, 2011.

[2] B. Chihani and D. Collange. A Multipath TCP model for ns-3 simulator. In *In Proc. of WNS3*, 2011.

[3] A. Ford, C. Raiciu, and M. Handley. TCP Extension for Multipath Operation with Multiple Addresses. In *RFC 6824*, 2013.

[4] S. F. Kevin Fall. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. In *In Proc. ACM SIGCOMM*, 1996.